Understanding Disruptive Monitoring Capabilities of Programmable Networks

Paul Chaignon, Kahina Lazri, Jerome Francois, Olivier Festor

Understanding Disruptive Monitoring Capabilities of Programmable Networks

Paul Chaignon*‡, Kahina Lazri*, Jérôme François†, and Olivier Festor†‡
*Orange Labs, France. Email: {firstname.lastname}@orange.com
†INRIA - Nancy Grand Est, France. Email: {firstname.lastname}@inria.fr
‡LORIA, University of Lorraine, France

***Abstract*—The design shift proposed by OpenFlow, with its simple stateless dataplane, initially contributed to the success of Software-Defined Networks. Its lack of state, however, prevents the implementation of many dataplane algorithms. Network applications must therefore offload stateful operations to the control plane, thereby increasing latency and limiting network scalability. Thus, recent research efforts centered on the addition of stateful properties to switches.**

In this paper, we discuss the impact of emerging programmable dataplane abstractions on network monitoring. In particular, we investigate the need for dataplane states in the design of scalable monitoring applications. We argue that these abstractions are ill-suited for software switches as they retain hardware-specific limitations. Furthermore, we analyse the impact of stateful dataplane designs on the control plane visibility of the network. Finally, we identify opportunities for improvement in the design of stateful software switches.

I. Introduction

Traditional computer networks consist of interconnected fixed-function devices implemented on Application-Specific Integrated Circuits (ASICs) for line-rate processing. They comprise switches and routers for forwarding as well as middleboxes for specific functionalities, such as access control, intrusion detection, and load-balancing. Forwarding devices implement distributed network protocols for the control plane. Thus, the control and forwarding planes (also called dataplane) are tightly coupled inside closed-source network devices, which complicates the deployment of new protocols. Furthermore, these devices expose limited vendor-specific control interfaces that hinder network configuration processes.

The Software-Defined Network (SDN) paradigm aims at decoupling the control plane from the dataplane. Forwarding devices are controlled by a separate, centralized entity, the controller, through open interfaces. While logically centralized, the controller may still be distributed for scalability and fault-tolerance. Applications interfaced with the controller implement the intelligence of the network, whether it be traffic engineering, access control, or anomaly detection.

The SDN decoupling logic arises from a simple observation. Whereas the dataplane benefits from a hardware implementation for line-rate processing, the control plane benefits from a more flexible multi-purpose implementation. Thus, with SDN, switches and routers become simple forwarding devices while commodity servers host the control plane.

The interface between the controller and switches was initially standardized by the OpenFlow protocol [1]. It advocates simple stateless switches exposed through a match-action abstraction. This abstraction is particularly well-suited for traffic engineering applications as they leverage the global view of the centralized controller and require few functionalities from SDN switches [2]. Applications that do not fit the stateless match-action abstraction must rely on the control plane. Control plane implementations, however, incur higher latencies and stress the control to dataplane path.

This limitation is particularly challenging for monitoring applications. They all require their own peculiar view of the network: flow statistics for heavy-hitter detection [3] and round trip times for TCP performance analysis [4], for example. Monitoring applications could summarize measurements at the dataplane, but the limited capabilities exposed by OpenFlow force switches to send large quantities of raw measurements to the controller.

The inadequacy of OpenFlow for network monitoring was recognized and discussed early on in [5], [6]. Enabling preprocessing of measurements in the dataplane requires persistent switch memories – referred to as the dataplane state – to store intermediate results. Recent proposals for new SDN paradigms explored how this dataplane state can be exposed to applications at the control plane.

In this paper, we investigate how these expositions of the dataplane state impact the design of monitoring applications. In particular, an unconstrained access to this state enables the implementation of any dataplane algorithm [7], [8], but introduces two main challenges. First, it limits the control plane's visibility, as any action taken based on the dataplane state becomes invisible to the controller. Second, an unconstrained access to the dataplane state is only possible on commodity hardware and thus, achieves two orders of magnitude lower performance than fixed-function ASIC switches. This second challenge was the subject of recent advances in hardware design, with the emergence of the first programmable ASIC switches with stateful processing capabilities [9]–[11].

The remainder of this paper is organized as follows. In section II, we detail the challenges for the implementation of monitoring algorithms. We then classify research proposals in section III according to the flexibility they offer for stateful dataplane implementations. In section IV, we analyze the implications for the control plane's global view. Finally, in section V, we discuss shortcomings in recent emerging stateful designs and identify new research opportunities.

II. Network Monitoring Challenges

Several scalability challenges for network monitoring applications in SDNs stem from dataplane resource limitations and affect both software and hardware switches [12].

A. Hardware Resource Limitations

Resource limitations in hardware switches are two-fold. First, flow tables in hardware switches have a limited size due to the use of Ternary Content-Addressable Memories (TCAMs). Switches use TCAMs to store and match against wild-card flow rules. These hardware components are expensive and consume considerable power and ASIC space. Thus, switch vendors often rely on less expensive and power-hungry Binary Content-Addressable Memories (BCAMs) and Static Random Access Memories (SRAMs) [12]. Second, hardware switches have limited CPU capacity, essentially used by auxiliary components, such as the OpenFlow switch agent.

Conversely, because software switches rely on commodity hardware, they dispose of large amounts of slower memories, DRAMs, with transparent memory caches on SRAMs. Servers can allocate enough DRAM space for the switch software to store hundred of thousands of rules. The CPU, however, is a limiting factor [13]. The number of CPU cycles needed to process one packet determines the switch's throughput and latency. To avoid context switches, operators often dedicate a core to the switch. Nevertheless, auxiliary functions of the switch, such as rule management, run on that same core and, as such, may impact performance.

B. Scalability Challenges

Hardware resource limitations introduce a trade-off between the granularity of measurements and the consumed resources. Because of this trade-off, monitoring large-scale networks is challenging, and network operators often resort to inaccurate sampling-based measurements.

The first scalability challenge stems from the strict binding between flow rules and counters and the resulting trade-off between the granularity of flow rules and the accuracy of statistics. For example, if an application defines flows based on their destination IP addresses, switches will report a single set of counters per destination IP address. Thus, an application that relies on fine-grained visibility of flows needs to install additional rules in switches.

The number of flow rules, however, is limited in both hardware and software switches. In hardware switches, wild-card rules are typically stored in TCAMs. As a consequence, while they support large numbers of exact-match rules, they can only match against a limited number of wildcard rules [5]. In software switches, there is no hard constraint on the number of flow rules. Nevertheless, with a high number of flow rules, the switch's CPU will spend more time managing rules, revalidating caches, and retrieving statistics for cached rules [13].

The second scalability challenge involves the frequency of statistic reports to the controller. A short delay between two statistic reports enables timely reaction to network events, but strains the path to the controller: the switch component that retrieves and sends statistics consumes CPU cycles, while the statistic reports themselves increase the network load. Preprocessing of statistics at the switch (e.g., aggregated reports) reduces the network overhead, but results in high CPU consumption.

III. Towards Programmable Network Monitoring

To address the scalability challenges for monitoring applications and the broader limitations of OpenFlow, several approaches have been proposed in the recent literature. This paper classifies these approaches according to the flexibility they offer for the implementation of dataplane algorithms.

We first focus on proposals for new control plane behaviors, as they preserve the stateless OpenFlow paradigm. We then discuss proposals to add a persistent state in the dataplane.

A. Control Plane-Based Monitoring

With the emergence of OpenFlow came proposals for software-defined network monitoring. The authors leverage the dynamicity and the global view over the network of OpenFlow to optimize resource utilization. They either consider a single switch and attempt to reduce its consumption of resources, or a coordinated set of switches to avoid local resource shortages.

1) Local Resource Optimization: Many applications are not interested in a fixed set of flows but rather in any flow with a specific characteristic. For example, to improve its efficiency, a traffic engineering application may monitor only elephant flows. Hence, to keep focus on relevant flows, dynamic adaptations of monitoring rules are required.

Jose *et al.* [3] propose to dynamically adapt the granularity of flow rules. They implemented a use case to identify the IP addresses of heavy hitters in several iterations. At each iteration, their prototype divides rules for flows containing heavy hitters into more precise sub-rules, thus increasing the granularity of flow rules. However, their incremental detection of heavy hitters takes a minimum of 200 seconds to converge, rendering this strategy inefficient for short-lived flows.

In PayLess [14], Chowdhury *et al.* propose a similar algorithm to adapt the reporting frequency, using a simple threshold-based algorithm. The algorithm increases the frequency if, over the last reporting period, the collected statistics changed more than a threshold Δ_1. Conversely, if the collected statistics changed less than a threshold Δ_2, the reporting frequency is decreased. Thus, PayLess monitors more closely highly variable flows. Chowdhury *et al.* demonstrate that the algorithm can halve the median reporting frequency while maintaining a high accuracy.

Y. Zhang proposes OpenWatch [15], an adaptive algorithm for anomaly detection. OpenWatch can update both the granularity of flow rules and the reporting frequency using a prediction-based algorithm. The linear prediction model estimates future statistics based on previous values. If the next collected values lie within the predicted interval, OpenWatch decreases the granularity (respectively the reporting frequency); otherwise, it increases the granularity.

Consequently, the algorithm allocates more resources on flows with an irregular behavior.

Propositions for an adaptive definition of flow rules only alleviate the scalability challenges. Thus, depending on the desired accuracy and the number of flows to monitor, a large number of flow rules and a high reporting frequency might still be needed. Furthermore, since these approaches add a convergence delay, they require temporal stability of monitored flows and may miss transient network events.

2) Network-Wide Resource Optimization: While resources might be scarce locally, large-scale networks are likely to contain many switches and thus, enough resources globally. This rationale motivates several approaches to distribute monitoring resources across the dataplane.

With Palette [16], Kanizo *et al.* design a flexible approach to the distribution of resources in the dataplane. Palette aims at balancing flow table sizes while minimizing the total number of flow rules. Because it acts directly on flow tables, it does not depend on the application logic.

The distribution of flow rules raises two underlying problems: the decomposition of flow tables into elementary flow tables and the placement of these elementary flow tables along network paths. Each flow table is decomposed such that a flow going through the succession of elementary flow tables will result in the same measurements as if it had gone through the original flow table.

In DCM [17], Yu *et al.* propose a greedy algorithm to distribute the monitoring load across switches. Contrary to Palette, DCM relies on a static exposition of state in switches, presented in section III-B. For this reason, DCM balances the memory usage across the dataplane, instead of the flow rules.

Similarly to local resource optimization approaches, proposals for network-wide optimization only alleviate the scalability challenges.

B. Monitoring Primitives in the Dataplane

The narrow measurement features of OpenFlow force switches to report large quantities of raw information to the controller. Hence, several proposed extensions for OpenFlow switches aim at processing measurements locally, before reporting the essential information to the controller. These approaches allocate a persistent state on switches to store intermediate measurements. These switch designs expose the persistent state through probabilistic data structures.

Probabilistic data structures, or sketches, are used by streaming algorithms to store summary information on the input data [18]. Sketches rely on hash algorithms to classify and count packets in a limited number of operations. As they store only the information required for the measurement task, they need few memory resources. Furthermore, these probabilistic data structures present a provable trade-off between the consumed resources and the achieved accuracy.

Probabilistic data structures have been extensively used to overcome resource limitations in the context of software-defined networking [6], [17], [19]. Indeed, they offer a better trade-off between resources and accuracy than counters, yet require little processing power.

In addition to the control plane algorithm for the distribution of rules presented in section III-A2, DCM [17] integrates a two-stage Bloom filter-based process for the definition of monitoring flow rules. The first Bloom filter, called the *admission Bloom filter*, selects flows to measure. Packets matching the admission Bloom filter are further processed in the second stage, which is composed of a Bloom filter, called *action Bloom filter*, per monitoring action. Packets are matched against each action Bloom filter to select the actions to apply. For example, one Bloom filter may lead to a Count-Min sketch, while another may forward packets to the controller, to implement sampling. This two-stage pipeline greatly decreases the overall number of false positives. Nevertheless, DCM is inadequate for some applications (e.g., stateful firewalls) for which even a small number of false positives is unacceptable.

Yu *et al.* design OpenSketch [6], a library and an API for traffic measurement, concurrent to OpenFlow. OpenSketch contains several probabilistic data structures, which measurement tasks can use as primitives. For example, a DDoS detection task would leverage Count-Min sketches, bitmaps and reversible sketches.

Complex measurement tasks, however, involve elaborate combinations of sketch primitives. Furthermore, each new measurement task requires custom algorithms and sketches. The absence of a sufficiently general solution hinders the development of sketches in production switches.

To solve this issue, in UnivMon [19], Liu *et al.* investigate the application of recent theoretical advances in universal streaming algorithms to network monitoring. Universal streaming algorithms can serve as the single primitive for a large number of monitoring tasks. Liu *et al.* develop a prototype and implement several measurement tasks from OpenSketch with this single data structure.

The authors of sketch-based proposals focused their evaluations on the accuracy and space constraints (memory, TCAMs, and number of hash functions). Only the authors of OpenSketch implemented it on a NetFPGA and measured the throughput, limited by the 1GbE link. Therefore, there is little information on whether sketch-based monitoring would scale to large networks, if implemented in hardware.

C. Stateful Software Dataplane

Counting and sampling methods are not always sufficient for monitoring applications, in particular for security applications. Applications need to monitor or take actions based on information from all network layers. For example, a denial-of-service detection system may track packets with a particular signature in their payloads to confirm an anomaly detected by flow counters.

These unique needs motivate Mekky *et al.* [8] and Sonchack *et al.* [7] to design OpenFlow extensions to run applications locally, in switches. Both approaches allow control plane applications to execute any custom program in switches. To this end, Mekky *et al.* add a rule in flow tables to redirect

unmatched packets to a separate table. The local application processes these packets and may forward them or alter flow tables as a result. Sonchack *et al.* adopt an alternate approach with OFX [7]: local applications control the first table in the OpenFlow pipeline and thus, they may intercept any packet independently of subsequent flow rules. As such, OFX keeps a strict separation between the flow table controlled locally and flow tables controlled by remote applications.

In essence, Mekky *et al.* and Sonchack *et al.* propose a local controller that can host applications in switches. Because such applications can only run on commodity hardware, this approach is limited to software and white-box switches and is inadequate for high throughput execution on ASIC.

Furthermore, the high flexibility of the local application approach comes with two barriers to deployment.

First, the processing of packets by local applications impede the controller's view of the dataplane. Even though the local applications are installed on switches by the controller, it remains unaware of any packet processing they perform. This is problematic for network verification applications as they rely on the omniscient control plane.

Second, neither Mekky *et al.* nor Sonchack *et al.* present solutions to isolate resources for local applications from resources for the OpenFlow forwarding pipeline. Hence, the controller cannot guarantee that a local application will not have a negative impact on packet forwarding. In particular, since the OpenFlow management agent and the local application compete for CPU resources, it may impact the switch ability to report statistics or forward packets to the remote controller.

D. Stateful Hardware Dataplane

Several recent efforts have focused on designing programmable dataplane hardware devices on ASIC for high performance networks. Most of these programmable dataplane designs expose persistent memory on switches, but differ in how it can be accessed and updated.

OpenState [20] models switches as finite state machines, where each packet corresponds to a transition. Therefore, it exposes the persistent switch memory through a finite number of states only. Because this abstraction appeared insufficient to implement many networking algorithms, FAST [21] and OPP [22] propose to associate a set of registers to the finite state machines. While FAST received a software evaluation only, the authors of OPP implemented their prototype on FPGAs. Even though they propose a departure from the match-action abstraction, they discuss how OPP can be mapped into match-action tables to leverage TCAMs and current ASIC hardware. Yet, they do not evaluate the potential increase in size of match-action tables to store all state machine transitions.

Recently, Bosshart *et al.* and Sivaraman *et al.* proposed to extend the match-action abstraction with stateful actions in P4 [23] and Domino [24] respectively. They expose the switch persistent memory as arrays, with a few restrictions on accesses to sustain line-rate hardware implementations. For example, they prohibit variable-sized loops, thus preventing the implementation of dynamic data structures, such as hash tables, linked lists, or priority queues. Furthermore, to allow for efficient pipelining, P4 and Domino also restrict the number of memory accesses to one per array and per table, forcing developers to use several arrays to implement a Bloom filter for example.

Recent proposals leverage the new stateful dataplane designs for network monitoring. Using P4, Ghasemi *et al.* implement Dapper [4], a TCP performance analysis system. Dapper inspects TCP sessions in real time near end-hosts and determines the origin of performance drops between the sender, the receiver, and the network itself. Dapper stores TCP sessions in a hash table, but, due to P4's lack of variable-sized loops, collisions are not resolved. The authors evaluate how larger hash tables can help reduce the probability of collisions.

Sivaraman *et al.* design HashPipe [25], a heavy hitter detection algorithm for programmable switches. They adapt a deterministic heavy hitter detection algorithm, which uses a hash table to store flow counts, for an implementation on P4. To avoid variable-sized lookups, they examine only a small, constant number of random items in the hash table. For this reason, HashPipe admits false positives, whose rate depends on the number of items examined and the size of the hash table.

The restrictions of current programmable hardware devices, in particular the lack of variable-sized loops, force developers to implement their own sketches. Although these language constraints stem from their line-rate hardware targets, they induce complex implementations, with higher memory consumption, reduced accuracy, and limited processing rates due to the use of additional tables.

IV. Impact on the Control Plane Visibility

One of the strength of the initial OpenFlow proposal is the controller's global visibility over the network. In particular, given the headers of a packet, the controller knows the path the packet will take through the dataplane and the actions it will be submitted to. This knowledge is leveraged by network verification applications [27], for example, to check invariants in the configuration of the dataplane (absence of loops, enforcement of ACL rules, etc.). With new stateful switches, however, any action dependent on dataplane state would be invisible to the controller. In this section, we discuss the impact of stateful dataplane designs on the controller's visibility over the network.

Table I presents the proposals previously discussed with the state properties induced for the dataplane and the impact on the controller's visibility. Column *Controller's visibility* refers to the minimum information the controller has on a packet, regardless of the dataplane algorithm. Within the *Dataplane state* column, *Stateful processing* refers to the capacity of the dataplane state to influence the actions taken on a packet, whereas, with *stateful monitoring* proposals, the dataplane state is used for preprocessing of measurements only. Therefore, contrary to stateful monitoring, stateful processing enables switches to act on a packet differently depending on past packets.

Category	Year	Proposal	Short description	Controller's visibility	Dataplane state
Control Plane-Based Monitoring	2011	Jose *et al.* [3]	Adapts granularity for heavy hitters detection	Path + actions	Stateless
	2013	OpenWatch [15]	Adapt the granularity and the reporting frequency based on a prediction algorithm	Path + actions	Stateless
	2013	Palette [16]	Balances flow table sizes across switches	Path + actions	Stateless
	2014	PayLess [14]	Adapts the reporting frequency	Path + actions	Stateless
	2014	OpenSample [26]	Sampling based measurement platform	Path + actions	Stateless
Monitoring Primitives in the Dataplane	2013	OpenSketch [6]	Separates measurement pipeline based on sketches	Path + actions	Stateful monitoring
	2014	DCM [17]	Distributed selection of flows to monitor based on Bloom filters	Path + actions	Stateful monitoring
	2015	UnivMon [19]	Universal sketches as the single primitive for measurement tasks	Path + actions	Stateful monitoring
Stateful Software Dataplane	2014	Mekky *et al.* [8]	Local controller in switches to implement dataplane applications	None	Stateful processing
	2015	OFX [7]	Extension to load custom applications in switches	None	Stateful processing
Stateful Hardware Dataplane	2014	OpenState [20]	Models packet forwarding with finite state machines	Possible actions	Limited stateful processing
	2014	FAST [21]	Adds register memories to finite state machines	Possible actions	Stateful processing
	2015	P4 [22]	Programmable switch language with memories as arrays	None	Stateful processing
	2016	OPP [22]	Adds register memories to finite state machines and discusses mapping to TCAMs	Possible actions	Stateful processing
	2016	Domino [22]	Programmable switch and language with line-rate processing guarantee	None	Stateful processing

TABLE I: Comparison of impact on dataplane state and controller's visibility of each proposal

Because they preserve the OpenFlow abstractions, all proposals for new control plane behaviors retain the controller's omniscience. Conversely, proposals that enable stateful processing in the dataplane limit the controller's visibility over the network.

In particular, with P4, Domino, and local applications, dataplane algorithms may be defined such that the controller has no knowledge on the actions applied on a packet at runtime. For example, a P4 action may read the egress port from an array in the persistent memory. Since the controller has no visibility over the switch memory, it cannot know on which port the packet is sent out either.

Interestingly, not all proposals for stateful processing in the dataplane result in the same reduction of the controller's visibility. Whatever the state machines installed in the dataplane, with OPP, OpenState, and FAST, the controller can always determine the finite set of possible actions applied on a packet. The difference with other stateful processing designs resides in the inability to define stateful actions. With state machine-based approaches, actions depend on state machine transitions, but are chosen from a static set of predefined actions. Conversely, with P4 and Domino, actions can be defined at runtime, from the dataplane state.

Finally, sketch-based proposals for dataplane design retain the full controller's visibility. Indeed, because they advocate the use of a second stateful pipeline dedicated to monitoring, they preserve the initial stateless packet processing pipeline of OpenFlow. The absence of negative impact on the controller's visibility is thus a direct consequence of the use of the dataplane state for preprocessing of measurements only.

V. Discussion & Opportunities

P4 has recently gained momentum with the emergence of the first programmable P4 switch on the market [9]. Support for P4 is also expected in NICs and software switches. As Dapper [4] and HashPipe [25] demonstrate, the presence of stateful programmable devices in networks would enable the development of new monitoring applications for use in large-scale networks.

The widespread deployment of P4 network devices, however, will largely depend on the availability of middle-end devices. Furthermore, we do not know yet to what extent such devices will follow the P4 specifications. For example, the lack of persistent general-purpose memories would make P4 less attractive to monitoring applications.

Moving forward, we identify three open research issues and opportunities for improvement in the design of dataplane devices.

First, the impact of stateful dataplane designs on network visibility has been overlooked by the research community thus far. New programmable switches could benefit from discussion on what an acceptable loss of visibility would be. Furthermore, network verification applications will require new methods and tools to overcome the loss of visibility and analyze the impact of dataplane states.

Second, we argue that their is still room for improvement in the design of stateful software switches. The first languages for programmable devices were designed for hardware switches [23], [24]. For this reason, they retain hardware specific limitations (discussed in section III-D). However, because software switches run on CPUs, restrictions on the access to persistent memories are unnecessary.

Furthermore, because software switches typically aim for lower performance than ASIC switches, accessing dynamic data structures at line-rate is feasible. Thus, software switches could expose their persistent memory through hash table structures, for example, instead of static data structures, such as arrays in P4. This new abstraction for the dataplane state would enable new use cases. For example, the implementation of a stateful firewall without relying on a controller requires a dynamic data structure. Indeed, because a stateful firewall needs to keep information on a variable number of established

connections in memory, it cannot rely on a fixed size array. Moreover, as is the case for many security applications, probabilistic data structures are ill-suited because even a small number of false positives is unacceptable (allows illegitimate connections through).

Third, current specifications for programmable devices detail how processing pipelines can be defined, but not how they can be updated. The P4 specification [23], for example, gives no indication on how to update the configuration of the device once it is running. This information is particularly important for network monitoring. The ability to load stateful algorithms in the switch at runtime, without interruption of packet processing, would enable the execution of monitoring algorithms for short periods of time. Network operators could use it for debugging or to gather additional information on network events of interest. This feature is easier to implement in software switches, as the switch must maintain two concurrent processing pipelines for a short time to avoid an interruption of packet processing.

VI. Conclusion

In this paper, we present recent work on network monitoring in software-defined networking. Network monitoring long called for the exposition of a dataplane state. The recent emergence of stateful programmable network devices may represent a fundamental shift for the design of monitoring applications. If these new devices live up to expectations, they will enable the development of new, unpredictable applications and their deployment in large-scale networks.

We discussed the impact of stateful dataplane designs on the control plane visibility over the network. Leveraging our analysis of requirements for monitoring applications, we identified three opportunities for improvement in the design of stateful network devices. In particular, for software switches, there is an opportunity to design a stateful programmable abstraction better suited to their commodity hardware environment.

Acknowledgments

We would like to thank Sylvie Laniepce for her helpful comments on drafts of this paper.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, and J. Zhou, "B4: Experience with a Globally-Deployed Software Defined WAN," *Proc. ACM SIGCOMM 2013 Conf.*, p. 3.

[3] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. 11th USENIX Conf. Hot Topics in Management Internet, Cloud, and Enterprise Networks and Services*, Berkeley, CA, USA, 2011, pp. 13–18.

[4] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data Plane Performance Diagnosis of TCP," *ArXiv e-prints*, Nov. 2016.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM 2011 Conf.*, pp. 254–265.

[6] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. 10th USENIX Conf. Networked Systems Design and Implementation*, Berkeley, CA, USA, 2013, pp. 29–42.

[7] J. Sonchack, J. M. Smith, A. J. Aviv, and E. Keller, "Enabling practical software-defined networking security applications with OFX," in *23nd Annu. Network and Distributed System Security Symp., NDSS, San Diego, CA, USA, Feb. 21-24, 2016*.

[8] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in sdn," in *Proc. 3rd Workshop Hot Topics in Software Defined Networking*, 2014, pp. 13–18.

[9] Barefoot Networks. Barefoot Tofino. [Online]. Available: https://www.barefootnetworks.com/technology/#tofino

[10] Intel Ethernet Switch FM6000 Series. [Online]. Available: http://www.intel.com/content/www/us/en/ethernet-products/switch-silicon/ethernet-switch-fm5000-fm6000-series.html

[11] P. Bosshart, G. Gibb, H.-s. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," *Proc. ACM SIGCOMM 2013 Conf.*, p. 99.

[12] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics in Software Defined Networking*, 2013, pp. 73–78.

[13] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks, and M. Casado, "The Design and Implementation of Open vSwitch," *12th USENIX Symp. Networked Systems Design and Implementation*, pp. 117–130, 2015.

[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE Network Operations and Management Symp. (NOMS)*, 2014.

[15] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proc. 9th ACM Conf. Emerging Networking Experiments and Technologies*, 2013, pp. 25–30.

[16] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM, 2013 Proc. IEEE*, pp. 545–549.

[17] Y. Yu, C. Qian, and X. Li, "Distributed and collaborative traffic monitoring in software defined networks," in *Proc. 3rd Workshop Hot Topics in Software Defined Networking*, 2014, pp. 85–90.

[18] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, p. 323, 2002.

[19] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," *Proc. ACM SIGCOMM 2016 Conf.*, no. Question 24, pp. 101–114.

[20] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.

[21] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for SDN," *Proc. ACM SIGCOMM 2014 Conf.*, no. 1, pp. 377–378.

[22] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *ArXiv e-prints*, May. 2016.

[23] The P4 Language Consortium, "The P4 Language Specification," 2015.

[24] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet Transactions: High-Level Programming for Line-Rate Switches," *Proc. ACM SIGCOMM 2016 Conf.*, pp. 15–28.

[25] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Smoking Out the Heavy-Hitter Flows with HashPipe," *ArXiv e-prints*, Nov. 2016.

[26] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *Proc. 34th IEEE Int. Conf. Distributed Computing Systems*, 2014, pp. 228–237.

[27] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying Network-Wide Invariants in Real Time," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, p. 467, 2012.